Name: Vanshika Ambwani

Roll No: 01

Subject: Blockchain and DLT

Experiment No.: 2

Aim: Create a Blockchain using Python

1. What is Blockchain?

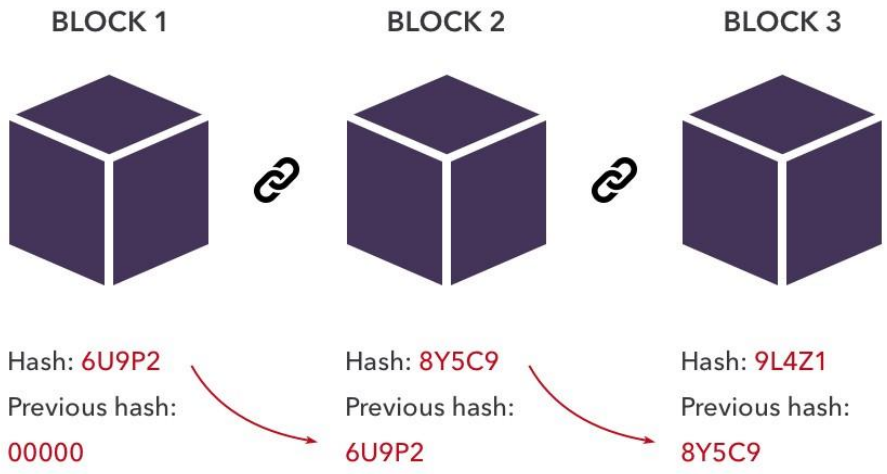
Blockchain is a distributed digital ledger technology used to store information in a secure and transparent manner. It is called “blockchain” because the data is stored inside blocks, and these blocks are connected together in the form of a chain.

Each block contains a set of records (such as transactions), and once a block is added to the chain, it becomes very difficult to modify or delete the stored data. This makes blockchain highly secure and reliable.

Blockchain works on a peer-to-peer network, meaning multiple computers (called nodes) maintain the same copy of the blockchain. Instead of relying on a single central authority (like a bank), blockchain ensures trust using cryptography, consensus mechanisms, and validation rules.

Key Features of Blockchain

- Decentralized: No single authority controls the data.
- Transparent: Transactions can be verified by all participants.
- Immutable: Once added, data cannot be easily changed.
- Secure: Uses cryptographic hashing and digital signatures.



2. What is a Block?

A block in a blockchain is a digital container or container acting as a permanent, immutable record for storing data, most commonly transaction information. It functions as a single unit in the chain, containing a list of validated transactions, a timestamp, and a unique cryptographic hash of the previous block.

3. Components of a Block

A typical block in a blockchain contains the following components:

- Block Header: Contains metadata used to identify and verify the block.
    - ○ Previous Block Hash: Links the current block to the preceding one, ensuring the integrity of the chain.
    - ○ Merkle Root: A cryptographic hash of all transactions in the block, enabling efficient validation.
    - ○ Timestamp: Records the precise time the block was created.
    - ○ Nonce: A number used in mining to satisfy proof-of-work puzzles.
    - ○ Version: Indicates the set of block validation rules to follow.
- Block Body: Contains the actual transaction data, such as sender/receiver addresses, transaction amounts, and smart contract execution details.
- Block Size: Defines the maximum amount of data (e.g., transactions) the block can store.
-

Why these components matter

- Hashing ensures security.
- Previous hash ensures linking.
- Merkle root ensures transaction integrity.
- Nonce supports mining and proof-of-work.

4. Process of Mining

Mining is the process of adding a new block to the blockchain by solving a complex mathematical puzzle. It is mainly used in **Proof of Work (PoW)** based blockchains such as Bitcoin.
Mining ensures that transactions are verified and prevents fraud like double spending.

Steps of Mining

1. **Transaction Collection**
   New transactions are collected from the network into a pool (mempool).

2. **Block Formation**
   Miners create a candidate block by adding transactions and block header details.

3. **Hash Calculation**
   The miner generates a hash of the block using cryptographic algorithms (example: SHA-256).

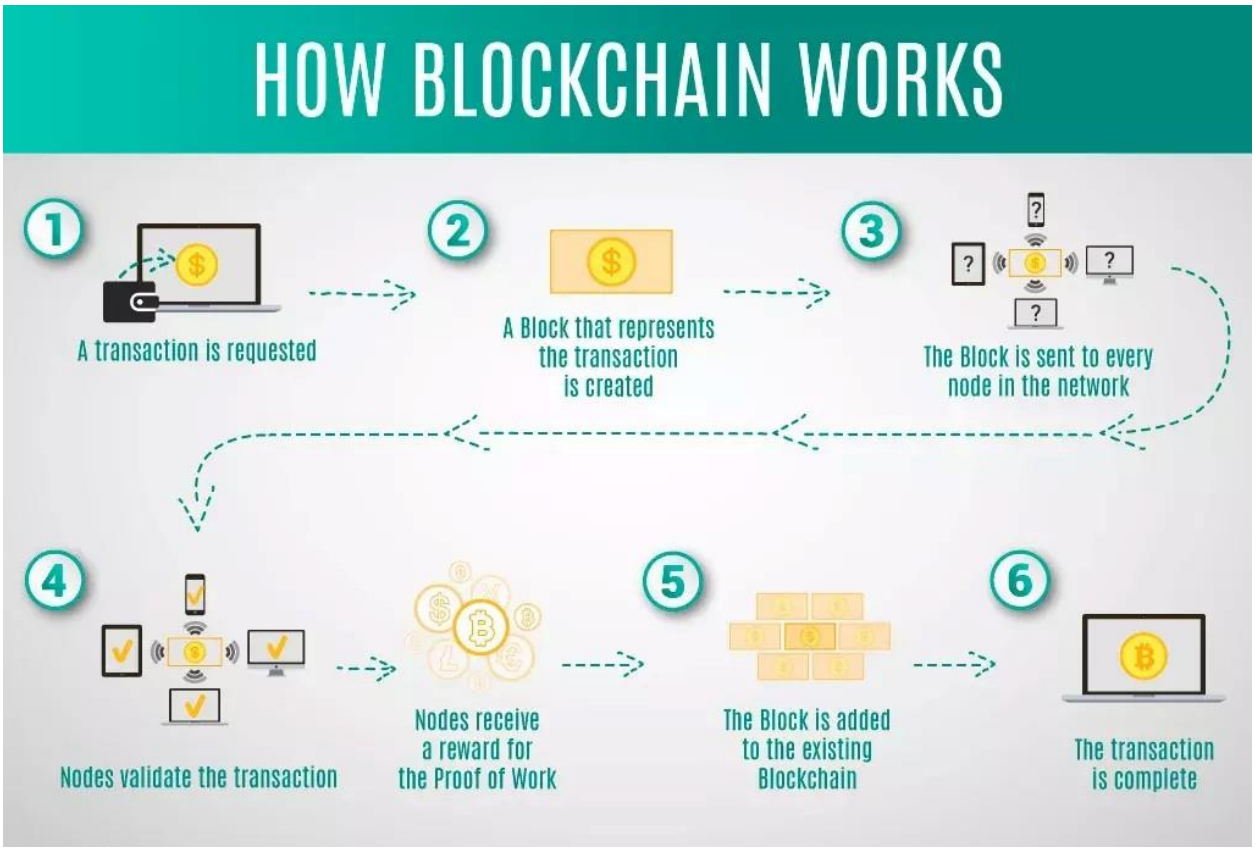
4. **Finding the Nonce (Puzzle Solving)**
   Miners repeatedly change the nonce to find a hash that meets the network difficulty target.

5. **Block Verification**
   Once a miner finds a valid hash, the block is broadcast to the network.

6. **Block Added to Blockchain**
   Other nodes validate it, and then it becomes part of the blockchain permanently.

Why Mining is Important

- Confirms valid transactions
- Secures the network
- Adds new blocks to blockchain
- Maintains trust without a central authority



5. How to Check the Validity of Blocks in Blockchain

To ensure blockchain security, every block is verified before being accepted. Block validity is checked using multiple validation rules.

Methods to Check Block Validity

(A) Hash Verification

Each block has its own unique hash.
If any data inside the block changes, the hash changes completely.
So, the system checks whether the hash is correct.

(B) Previous Hash Matching

Every block contains the hash of the previous block.
To validate a block:

- The previous hash stored in the current block must match the hash of the previous block.

(C) Proof of Work Validation

In PoW blockchains, nodes check whether the miner has solved the puzzle correctly.
This is verified by checking if the block hash satisfies the difficulty level.

(D) Merkle Root Verification

Merkle root ensures all transactions are correct.
If any transaction is modified, the Merkle root changes, and the block becomes invalid.

(E) Consensus Mechanism

A block is considered valid only if the majority of network nodes agree.
This agreement is called consensus.

Code:

```
import datetime     # To generate timestamps for blocks
import hashlib      # To generate SHA256 hashes
import json         # To convert block data into JSON
from flask import Flask, jsonify    # To create API endpoints
```

```


# ------------------------------------------------------------
# Part 1 - Building the Blockchain
# ------------------------------------------------------------


class Blockchain:


    def __init__(self):
        # This list will store the entire chain of blocks
        self.chain = []
```

```
    # Create the genesis block (first block)
    # proof = 1 → arbitrary
    # previous_hash = '0' → since no previous block exists
    self.create_block(proof=1, previous_hash='0')

def create_block(self, proof, previous_hash):
    """
    Creates a new block and adds it to the chain.
    Each block contains:
    - index
    - timestamp
    - proof (PoW output)
    - previous block hash
    """
    block = {
        'index': len(self.chain) + 1,            # Position of block in chain
        'timestamp': str(datetime.datetime.now()), # Current timestamp
        'proof': proof,                          # PoW result
        'previous_hash': previous_hash           # Hash of the previous block
    }

    # Add block to the chain
    self.chain.append(block)
    return block
```

```
    def
get_previous_block(self):
        """
        Returns the last block in the chain.
        """
        return self.chain[-1]


    def proof_of_work(self, previous_proof):
        """
        Simple Proof of Work (PoW) algorithm:
        - Find a number (new_proof)
        - Such that the SHA256 hash of (new_proof^2 - previous_proof^2)
starts with '0000'


        This is a computational puzzle to secure the network.
        """
        new_proof = 1
check_proof = False


        while check_proof is False:
            # Cryptographic puzzle: hash difference of squares
```

```
            hash_operation = hashlib.sha256(
                str(new_proof**2 - previous_proof**2).encode()
            ).hexdigest()
```

```
        # Check if hash begins with 4 leading zeros
        if hash_operation[:4] == '0000':
            check_proof = True
        else:
            new_proof += 1

    return new_proof

  def hash(self, block):
    """
    Creates a SHA256 hash of a block.
    - Sort keys to ensure consistent hashing
    """
    encoded_block = json.dumps(block, sort_keys=True).encode()
    return hashlib.sha256(encoded_block).hexdigest()

  def is_chain_valid(self, chain):
    """
    Validates the blockchain by checking:
1. The previous_hash field matches the actual hash of the previous block.
2. The PoW condition (hash starting with '0000') is satisfied for each block.
    """
    previous_block = chain[0]   # Genesis block
    block_index = 1           # Start checking from block 2
```

```
        while block_index < len(chain):
            block = chain[block_index]

            # Check previous hash correctness
            if block['previous_hash'] != self.hash(previous_block):
                return False

            # Validate Proof of Work
            previous_proof = previous_block['proof']
            proof = block['proof']
            hash_operation = hashlib.sha256(
                str(proof**2 - previous_proof**2).encode()
            ).hexdigest()

            if hash_operation[:4] != '0000':
                return False

            # Move to next block
previous_block = block
            block_index += 1

        return True



# -----------------------------------------------------------
```

```
# Part 2 - Mining our Blockchain (Flask API)
# ------------------------------------------------------------
```

```

# Initialize Flask web application
app = Flask(__name__)

# Create an instance of the Blockchain
blockchain = Blockchain()

# ------------------------------------------------------------
# API Endpoint: Mine a new block
# ------------------------------------------------------------
@app.route('/mine_block', methods=['GET'])
def mine_block():
    # Get the previous block
    previous_block = blockchain.get_previous_block()

    # Extract previous proof
    previous_proof = previous_block['proof']

    # Run Proof of Work algorithm
    proof = blockchain.proof_of_work(previous_proof)

    # Get hash of previous block
    previous_hash = blockchain.hash(previous_block)
```

```
    # Create the new block
    block = blockchain.create_block(proof, previous_hash)

    # Prepare response
    response = {
        'message': 'Congratulations, you just mined a block!',
        'index': block['index'],
        'timestamp': block['timestamp'],
        'proof': block['proof'],
        'previous_hash': block['previous_hash']
    }
    return jsonify(response), 200


# ----------------------------------------------------------
# API Endpoint: Get the full blockchain
# ----------------------------------------------------------
@app.route('/get_chain', methods=['GET'])
def get_chain():
    response = {
        'chain': blockchain.chain,
        'length': len(blockchain.chain)
    }
    return jsonify(response), 200
```

```
# ------------------------------------------------------------
# API Endpoint: Check if blockchain is valid
# ------------------------------------------------------------
@app.route('/is_valid', methods=['GET'])
def is_valid():
    is_valid = blockchain.is_chain_valid(blockchain.chain)

    if is_valid:
        response = {'message': 'All good. The Blockchain is valid.'}
else:
        response = {'message': 'Houston, we have a problem. The Blockchain is not
valid.'}

    return jsonify(response), 200



# ------------------------------------------------------------
# Run the Flask app
# ------------------------------------------------------------
app.run(host='0.0.0.0', port=5000)
```

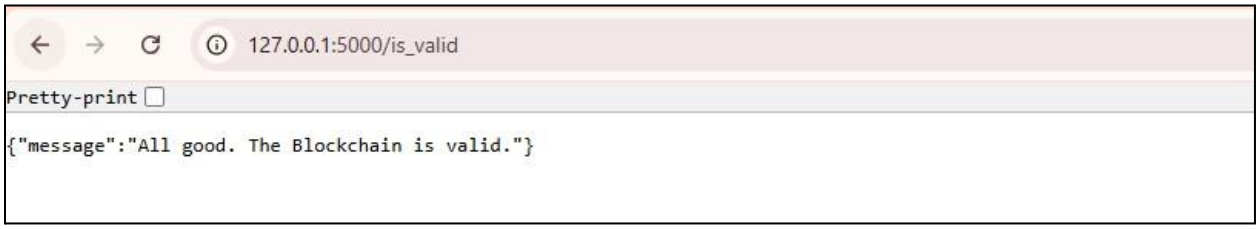
Output:

```
C:\Users\INFT507-10>pip install flask==2.2.5
Defaulting to user installation because normal site-packages is not writeable
Collecting flask==2.2.5
  Downloading Flask-2.2.5-py3-none-any.whl.metadata (3.9 kB)
Collecting Werkzeug>=2.2.2 (from flask==2.2.5)
  Downloading werkzeug-3.1.5-py3-none-any.whl.metadata (4.0 kB)
Collecting Jinja2>=3.0 (from flask==2.2.5)
  Downloading jinja2-3.1.6-py3-none-any.whl.metadata (2.9 kB)
Collecting itsdangerous>=2.0 (from flask==2.2.5)
  Downloading itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9 kB)
Collecting click>=8.0 (from flask==2.2.5)
  Downloading click-8.3.1-py3-none-any.whl.metadata (2.6 kB)
Collecting colorama (from click>=8.0->flask==2.2.5)
  Downloading colorama-0.4.6-py2.py3-none-any.whl.metadata (17 kB)
Collecting MarkupSafe>=2.0 (from Jinja2>=3.0->flask==2.2.5)
  Downloading markupsafe-3.0.3-cp313-cp313-win_amd64.whl.metadata (2.8 kB)
Downloading Flask-2.2.5-py3-none-any.whl (101 kB)
Downloading click-8.3.1-py3-none-any.whl (108 kB)
Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Downloading jinja2-3.1.6-py3-none-any.whl (134 kB)
Downloading markupsafe-3.0.3-cp313-cp313-win_amd64.whl (15 kB)
Downloading werkzeug-3.1.5-py3-none-any.whl (225 kB)
Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Installing collected packages: MarkupSafe, itsdangerous, colorama, Werkzeug, Jinja2, click, flask
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6/7 [flask]  WARNING: The script flask.exe is installed in 'C:\Users\INFT507
-10\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.13_qbz5n2kfra8p0\LocalCache\local-packages\Python313\Script
s' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
```

```
C:\Users\INFT507-10>C:\Users\INFT507-10\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.13_qbz5n2k
fra8p0\python.exe -m pip install --upgrade pip
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pip in c:\program files\windowsapps\pythonsoftwarefoundation.python.3.13_3.13.2544.0_x64_
_qbz5n2kfra8p0\lib\site-packages (25.2)
Collecting pip
  Downloading pip-25.3-py3-none-any.whl.metadata (4.7 kB)
Downloading pip-25.3-py3-none-any.whl (1.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 2.7 MB/s  0:00:00
Installing collected packages: pip
  WARNING: The scripts pip.exe, pip3.13.exe and pip3.exe are installed in 'C:\Users\INFT507-10\AppData\Local\Packages\Py
thonSoftwareFoundation.Python.3.13_qbz5n2kfra8p0\LocalCache\local-packages\Python313\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed pip-25.3
```

```
C:\Users\INFT507-10>python blockchain.py
 * Serving Flask app 'blockchain'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.33.32:5000
Press CTRL+C to quit
127.0.0.1 - - [21/Jan/2026 22:52:09] "GET /mine_block HTTP/1.1" 200 -
127.0.0.1 - - [21/Jan/2026 22:52:11] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [21/Jan/2026 22:52:16] "GET /mine_block HTTP/1.1" 200 -
127.0.0.1 - - [21/Jan/2026 22:53:42] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [21/Jan/2026 22:54:45] "GET /is_valid HTTP/1.1" 200 -
127.0.0.1 - - [21/Jan/2026 22:55:27] "GET /mine_block HTTP/1.1" 200 -
127.0.0.1 - - [21/Jan/2026 22:55:28] "GET /mine_block HTTP/1.1" 200 -
127.0.0.1 - - [21/Jan/2026 22:55:32] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [21/Jan/2026 22:55:38] "GET /is_valid HTTP/1.1" 200 -
```

{"chain":[{"index":1,"previous_hash":"0","proof":1,"timestamp":"2026-01-21 22:50:16.230188"},
{"index":2,"previous_hash":"3c1360299905075aaf4149cc55e6d6e4963332d677b55eb06948445cd840fd0a","proof":533,"timestamp":"2026-01-21 22:52:09.813081"},
{"index":3,"previous_hash":"11a8d4465faa1817d23c6acb700d8b13a7ea0f52b1f6ca6a2922d3f0f52af3cd","proof":45293,"timestamp":"2026-01-21 22:52:16.052502"},
{"index":4,"previous_hash":"edf0f1b2203fd1b811d4f454ef169a93e14ae07f691fe64f5df2d8203bf61ef5","proof":21391,"timestamp":"2026-01-21 22:55:27.667984"},
{"index":5,"previous_hash":"dec61000a4b0105cfda01a7931c9ad3530366fb8ad4d9eb9d7c538f02979e34e","proof":8018,"timestamp":"2026-01-21 22:55:28.682613"}],"length":5}



{"message":"All good. The Blockchain is valid."}